Recursion cheatsheet for coding interviews

Introduction

Recursion is a method of solving a computational problem where the solution depends on solutions to smaller instances of the same problem.
All recursive functions contains two parts:

1. A base case (or cases) defined, which defines when the recursion is stopped - otherwise it will go on forever!
2. Breaking down the problem into smaller subproblems and invoking the recursive call

One of the most common example of recursion is the Fibonacci sequence.

- Base cases: `fib(0) = 0` and `fib(1) = 1`
- Recurrence relation: `fib(i) = fib(i-1) + fib(i-2)`

```
def fib(n):
 if n <= 1:
   return n
 return fib(n - 1) + fib(n - 2)
```

Many algorithms relevant in coding interviews make heavy use of recursion - binary search, merge sort, tree traversal, depth-first search, etc. In this article, we focus on questions which use recursion but aren't part of other well known algorithms.

Learning resources

- Readings
    - Recursion, University of Utah
- Videos
    - Tail Recursion, University of Washington

Things to look out for during interviews

- Always remember to always define a base case so that your recursion will end.
- Recursion is useful for permutation, because it generates all combinations and tree-based questions. You should know how to generate all permutations of a sequence as well as how to handle duplicates.
- Recursion implicitly uses a stack. Hence all recursive approaches can be rewritten iteratively using a stack. Beware of cases where the recursion level goes too deep and causes a stack overflow (the default limit in Python is 1000). You may get bonus points for pointing this out to the interviewer. Recursion will never be O(1) space complexity because a stack is involved, unless there is tail-call optimization (TCO). Find out if your chosen language supports TCO.
- Number of base cases - In the fibonacci example above, note that one of our recursive calls invoke `fib(n - 2)`. This indicates that you should have 2 base cases defined so that your code covers all possible invocations of the function within the input range. If your recursive function only invokes `fn(n - 1)`, then only one base case is needed

Corner cases

- `n = 0`
- `n = 1`
- Make sure you have enough base cases to cover all possible invocations of the recursive function

Techniques

Memoization

In some cases, you may be computing the result for previously computed inputs. Let's look at the Fibonacci example again. `fib(5)` calls `fib(4)` and `fib(3)`, and `fib(4)` calls `fib(3)`and `fib(2)`. `fib(3)` is being called twice! If the value for `fib(3)` is memoized and used again, that greatly

improves the efficiency of the algorithm and the time complexity becomes O(n).

Essential questions

These are essential questions to practice if you're studying for this topic.

- Generate Parentheses
- Combinations
- Subsets

Recommended practice questions

These are recommended questions to practice after you have studied for the topic and have practiced the essential questions.

- Letter Combinations of a Phone Number
- Subsets II
- Permutations
- Sudoku Solver
- Strobogrammatic Number II (LeetCode Premium)